TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

EMIL LAHTINEN

USING SESSION REPLICATION IN WEB SERVICES

Master Thesis

Examiner: Prof. Tommi Mikkonen
Examiner and topic approved by the
Faculty Council of the Faculty of
Computing and Electrical Engineering
on 3rd December 2014

ABSTRACT

EMIL LAHTINEN: Using session replication in web services
Tampere University of Technology
Master Thesis, 46 pages
March 2014
Master's Degree Programme in Information Technology
Major: Software Development
Examiner: Prof. Tommi Mikkonen
Keywords: NoSQL, web development, session replication

Web services have become to play an important role in people's everyday life. Along with their importance requirements and expectations for web services have risen and they will continue to grow in the future. Session replication is a technique which improves many quality aspects of web services, such as maintainability, availability and reliability. It reduces down time of the servers and betters the user experience by improving these factors in the wen service.

The idea in session replication is to make the web services individual application servers users session objects available for all the application servers by storing them on a third party storage.

In this thesis there is a survey on the technologies used in the web on a broad spectrum and a survey on the most promising open source session replication solution in the web. After studying and comparing these technologies, a pilot implementation is made to see the technologies in action and to see how they compare to a solution where no session replication is used. The goal is to find the best option for production system session replication.

All of the implemented replication technologies worked and the goals set for this thesis were met. The findings of this study will be used as a baseline for the production session replication solutions.

TIIVISTELMÄ

EMIL LAHTINEN: Sessioreplikoinnin hyödyntäminen verkkopalveluissa
Tampereen teknillinen yliopisto
Diplomityö, 46 sivua
Maaliskuu 2014
Tietotekniikan koulutusohjelma
Pääaine: Ohjelmistotuotanto
Tarkastajat: Prof. Tommi Mikkonen
Avainsanat: NoSQL, Web-sovelluskehitys, Sessioreplikointi

Verkkopalvelut ovat kehittyneet niin tärkeiksi, että ne näyttelevät suurta roolia ihmisten jokapäiväisessä elämässä. Tämä merkitys tulee vain kasvamaan tulevaisuudessa. Verkkopalvelujen tärkeyden myötä ovat kasvaneet myös odotukset ja vaatimukset palveluja kohtaan. Käyttämällä sessioreplikointia pystytään parantamaan verkkopalvelujen ylläpidettävyyttä, saatavuutta, sekä luotettavuutta ja täten osittain vastaamaan kasvaneita vaatimuksia.

Sessioreplikointi on tekniikka, jossa yksittäisen sovelluspalvelimen sessio-objektit tuodaan muiden sovelluspalvelimien tietoon tallentamalla kaikkien sovelluspalvelimien sessio-objektit yhteiseen paikkaan, johon jokaisella sovelluspalvelimella on yhteys. Tämä tallennuspaikka on toteuttavasta teknologiasta riippuen jokin NoSQL tietokanta tai NoSQL:n kaltainen tietovarasto.

Tässä diplomityössä luodaan yleinen katsaus webissä käytettäviin tekniikoihin ja teknologioihin sekä tutkitaan tarkemmin lupaavimpia sessioreplikoinnin mahdollistavia teknologioita. Lisäksi työssä toteutetaan sessioreplikointi testipalveluun kolmella lupaavimmalla teknologialla, joita verrataan nykyiseen toteutukseen, jossa sessioreplikointia ei käytetä. Työn tarkoituksena on löytää paras vaihtoehto käytettäväksi tuotantoympäristössä.

Työssä toteutetut järjestelmät toimivat hyvin ja työ täytti sille asetetut odotukset. Tämän työn tuloksia tullaan käyttämään pohjana tuotantokäyttöön tarkoitetuissa replikointiratkaisuissa.

PREFACE

This thesis work started as a research study for my work place Alma Mediapartners over a year ago. Personally I wanted to gain more knowledge about NoSQL databases and I want to thank everyone at work contributing in making this subject possible for me.

I want to thank my superior Antti Keskitalo for proofreading and giving me valuable advice and insight on the subject. My biggest thanks goes to my thesis supervisor Tommi Mikkonen for providing his expertise in guiding this thesis work.

This thesis turned out to be everything I expected in satisfying my thirst for knowledge on the matter and becoming a better developer. I hope the knowledge gained while doing this thesis will help me and others in the future to develop sustainable replication solutions.

6.2.2015 Emil Lahtinen

TABLE OF CONTENTS

1. INTRODUCTION

Web sites have grown from static HTML pages to automatically scaling complex applications with dynamic content. Currently the web is an important channel in marketing, business and social interactions, and customer requirements for web services have grown alongside with webs importance. Web services have to be available for customers at any given time anywhere they want, and web services have be able to serve the growing number of customers.

Increasing requirements and expectations, number of customers and general growth of requests to web services bring challenges to the development of web services. A single server is not able to handle the load of a popular web service anymore and new kind of architectures and work flows are needed to address this issue.

There are a number of different solutions to mitigate these issues. The servers could be geographically distributed so the physical distance to customers is as short as it can be or the web services could be built in a distributed manner. When built with distributed architecture the web service is actually located on multiple servers which take care of their dedicated duties in the web service. The above solutions can be combined, and the whole service could be mounted to automatically scaling cloud services located around the world.

Normally web services do not reveal their inner architecture and complexity to users but there can be indirect cues. For example during a production update the time it takes to turn off, update and restart every server in the service could take too long from the users point of view.

The general expectations towards web services have also grown. Accessibility and ease of use are important factors for users when they decide which service they want to use and these factors correlate straight to the popularity of the web service. To satisfy the users needs the service should be available at any given time with minimal downtime.

The goal of this thesis is to study different replication technologies, compare them, and implement a working and sustainable solution for sharing user sessions across

the application servers using replication. Session replication was piloted on Alma Mediapartners' Autotalli.com web service but eventually it will be used in every Alma Mediapartners' web service. Alma Mediapartners' is a Finnish company providing various web services for users such as Etuovi.com, Vuokraovi.com, and Autotalli.com. Alma Mediapartners web services are among the most popular services in their field in Finland.

Chapter 2 will focus on the general architecture of web services regarding this work and explore all the technologies and components related to this work. Chapter 3 will focus on the techniques and technologies used in session replication and discussion of the pros and cons of using session replication in web services. Chapter 3 there will also be an estimate of the feasibility of each solution from the case study web services point of view. Furthermore in chapter 3 there will be a comparison of the different technologies. The architecture of Autotalli.com web service where the session replication will be implemented is presented. Wicket framework which is used to build the application will also be covered in chapter 4 and the actual modifications needed to implement session replication are presented in chapter 4. Chapter 5 contains testing of the implemented technologies and the test results. Comparison of the differences before and after session replication is also in chapter 5. Chapter 6 contains the conclusions of the work and the estimate of the usefulness and added value from the session replication. Chapter 6 also contains the possible future development ideas derived from this work.

2. THE ARCHITECTURE OF WEB

In this chapter we examine how computers work in the web on a broad spectrum. The basic concepts and most popular techniques are presented on general level without diving too much into detail. The presented technologies and methods relate closely to the goal of this thesis since all of them are used in the web service which is piloting for session replication. Some of the issues introduced in this chapter relate to replication on more abstract level but are still important issues to talk about to give the reader a sense of the big picture.

2.1 Client-Server pattern

When the web started to grow and take place in everyday tasks the place to store and manage information started to shift from the users personal computer towards different kinds of web services. Before long the web was full of popular web services offering a diversity of storing, managing and leisure services which increased the traffic in the web. In order to serve this increasing number of users web services had to be built in a manner which is not tied to a certain amount of users and the client-server pattern was the perfect architecture for this. In his article Client-Server model Oluwatosin calls the client-server pattern as the architecture of the web [1].

The pattern is quite simple: the client, in a basic web service case the user's browser, sends a request using a certain protocol for the server which interprets the request and sends a proper response for it [2]. The most common protocol used in web services is the HTTP protocol, but the client-server pattern is not tied to any protocol and can be implemented to use any protocol there is for example FTP protocol for file sharing and the SMTP protocol which is used in various email programs.

Using the client-server pattern gives many benefits. The pattern enables easy distribution of resources because the resources are centrally located on the server to which every client has access to. By using the client-server pattern part of the work can also be delegated to the client program which saves the resources of the server. The client-server pattern also reduces data replication because the data is stored to the server from where client programs can use the resources instead of maintaining

a copy on every client computer [3, 1].

2.2 The Tier models

The client-server pattern can be divided into components where presentation, business logic and data management are separated. These patterns are called n-tier models and the responsibility areas are divided in different ways depending on the tier. [3, 1]

2.2.1 2-tier model

The 2-tier model is the most basic one. In 2-tier model there are two components: The client program and the server program which are illustrated in Figure 2.1. The client program has all the business logic and information how to represent the data from the server. The client is connected straight to the database which in this tier model is equivalent to server. The 2-tier model is also called thick client model since all the work is done on the client side. [3, 4]

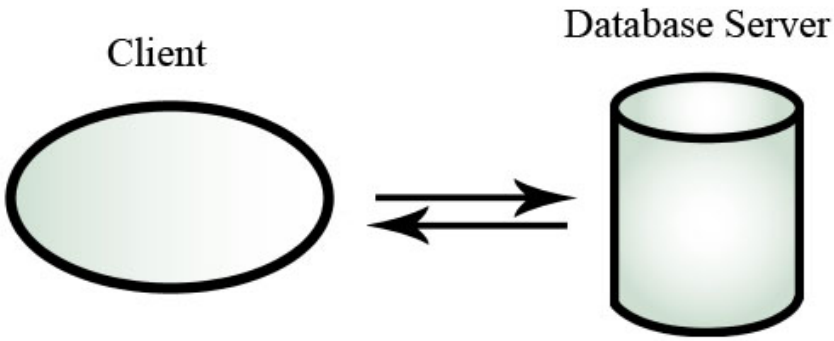

Figure 2.1 *2-tier model.*

The simple structure and easy maintenance are benefits of the 2-tier model but there are downsides when using this model as a web services architecture. In the 2-tier model the clients have direct access to the database. This jeopardizes information security of the database and leaves the database vulnerable to attacks because of the direct access. Also changes to business logic will not get updated automatically and need to be updated one client at a time. The 2-tier model is mostly used in intranets and closed environments, which have limited access and set of users. [3, 4]

2.2.2 3-tier model

More common and widely used web service architecture is the 3-tier model. The 3-tier model consists of a client program, an application server and a database server which are shown in Figure 2.2. In the 3-tier model the client program has only the representation logic for displaying the content served by the application server. In this model the application server serves as a middleware between the client and the database. All the business logic and management related to the information in the web service relies on the application server. The application server handles also the connection and communication to the database server. [3, 4]

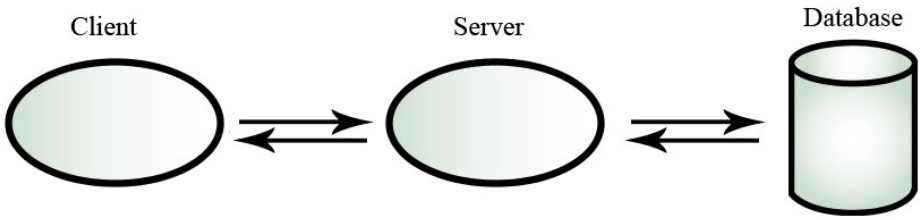

Figure 2.2 *3-tier model.*

The 3-tier model has many advantages when used as a web services architecture. Complex business logic can be implemented on the application server which enhances the efficiency of the database since the database has to primarily handle only the basic CRUD operations (create, read, update, delete). Changes in the business logic is easy to make because only the application server has to be updated. This makes updating easy and quick and the effects are seen on the client programs immediately. The 3-tier model also scales horizontally which means that more application servers can be added to handle the traffic without burdening the database. By adding more servers the model becomes the n-tier model. [3, 4]

The 3-tier model also has its downsides. The complexity of the whole system is increased because the system is made up from multiple different components. The increased complexity has its affects on the system maintenance. [3, 4]

2.3 Cookie

The most widely used protocol in the web, the HTTP protocol, is a stateless protocol. The server cannot distinguish requests coming from a certain user among all the

requests without some kind of management of state. From the servers perspective every request is its own totality and contains all the information needed to give the correct response. By using only a stateless protocol functionalities like a shopping cart would be very difficult to implement. [5, 6]

In the early days of the web cookies were created to address the statelessness problem. A cookie is a small amount of arbitrary data which is saved to the users browser. The cookie consist of six components: a name-value pair, the expiry time of the cookie, path the cookie is good for, domain the cookie is good for, is a secure connection needed to use the cookie and whether the cookie can be accessed by other means than HTTP, javascript for example. The server sends this cookie for to the users browser with the first response to be stored and used with the future requests. The user browser then stores it locally and sends it to the server with requests. [6, 7]

2.4 Session

By the use of cookies the management of interactions can be established between the web service and the user. This state is commonly called session. Maintaining a session is not the only application area of cookies. For example cookies can be used to personalize websites according to users previous settings by storing these settings in a cookie. The next time the user visits the website the server asks the browser for previous settings and tunes the appearance of the website with the users preferred settings. Also authentication for a website can be created using cookies. This is done by using cookies signed with a generated key based on users IP address, password or a digital signature. The name-value pair is required to be set explicitly. [8, 9, 10, 6]

One of the biggest and most controversial application area of cookies is user tracking. Websites from online shops to search engines are tracking users by monitoring which pages the user has been viewing and what kind of searches the user is doing on their website. By utilizing this knowledge of the users' behaviour, more specifically targeted advertisements can be shown to the user. [8, 9, 10, 6]

Most of the web services use sessions on their websites. When a user comes to the website the server creates a unique identifier for the user which the server saves to the users session and sends the identifier for the users browser where it is stored in a cookie. When the user makes a request to the server the cookie containing the identifier is also sent to the server from which the server can identify the user. This way the server can recognize and group all the requests the user has made during his visit to the website [11]. By using the information saved to the users session the

server can see what pages the user has been viewing, what information the user has been giving and what kind of searches the user has made.

Using cookies to store information about the user is relative easy and straightforward and for that reason cookies are widely used in web services. There are however downsides about using cookies. The biggest problem is that web services store information to the users browser without asking permission to install them. User privacy concerns regarding web services have been at the focus of many research studies and today the mainstream is to inform the users about the usage of cookies and to what purpose the web service is using cookies. [6, 12, 13, 8]

Another big problem related to cookies is session hijacking. If the user is identified only by a session identifier string stored to users browser it is relative easy to make a copy of the identifier and impersonate the user. This is a common method of session hijacking. [6, 12, 13, 8]

2.5 Load Balancing

The overall traffic has been growing drastically since the launch of the web, and popular web services in particular are getting a lot of traffic on their websites. The infrastructure has to be in a good shape to withstand the growing amount of traffic. Especially during special promotions and other user peaks the web services servers can get so much requests that they cannot respond to all of them. At this stage users trying to get to the website experience long loading times and probably will get ”Service temporarily unavailable” error. [14]

To avoid these situations one way is to run multiple instances of the application server which can linearly in relation to one instance serve more user than running just one application server assuming the use of servers with equal size of resources. Using multiple application servers is common in popular web services because it enables to serve more users and gives fault tolerance for the service. Running multiple servers however present a problem: the websites address can point only to one machine. This is where a load balancer is used. All the requests coming to the web service go through the load balancer which decides which application server is most capable to serve the request. [14]

Load balancing means the division of the workload to all available resources. In web services case it means distributing the incoming requests to all the available application servers evenly. Load balancing is needed to take full advantage of all the application servers capacity. [14]

Load balancing aims to optimize the use of resources, decrease response times, increase efficiency and to avoid overloading one single server. Using many instances of the application running on multiple servers has advantages over one server with massive resources because it increases accessibility and fault tolerance. Accessibility can be increased by distributing the physical servers amongst the users for example around Finland in a Finnish web service case. Fault tolerance and also accessibility are increased by load balancing because if one server crashes the other instances are still serving users. [14, 15]

2.6 Load balancing approaches

Load balancing can be handled with three different approaches: client-based approach, DNS-based approach and the server-based approach. All three satisfy the user transparency requirement. The requirement denotes that from the users point of view the mechanism is hidden or that the user does not need to be aware of the mechanism that executes this operation. The system setup is illustrated in Figure 2.3 [14]

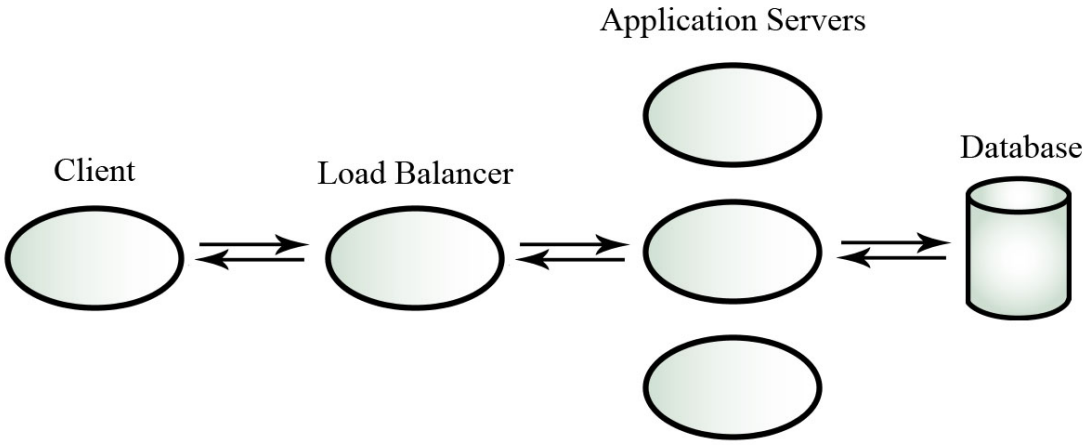

Figure 2.3 *Load Balancer.*

2.6.1 Client-based Approach

Load balancing can be implemented on the client side if we assume that the client programs know the existence of all the application servers on a web service. The client based approach is simple. When the user makes a request, the client is able to select a server to which it is going to submit the request. After submitting the client

needs to track that the appropriate server responses to this request. A successive request from the same client can reach another server in the service. Client side load balancing is mainly used in intranets and similar more restricted environments where the web service has some control over the client programs. This is because in the case where new servers are added to the service, every client program has to be updated for the load balancing to work properly. [14, 15]

2.6.2 DNS-based Approach

Another way to implement load balancing is the DNS approach but first we need to make a quick excursion to internet protocol. When a user types a network address in the browser such as *www.autotalli.com* the browser does not automatically know where it should submit the request. Instead it makes a request for what is known as a Domain Name Server and asks it to give the IP address that correlates with *www.autotalli.com*. An IP address is a numerical label assigned to each machine that is participating in a computer network that uses internet protocol for communication. In short term IP address is a unique address of a single machine in the internet. After the browser gets the IP address of the web service the user wanted it can make the request for the appropriate service. [14]

In a case of a clustered web service the service usually has a cluster DNS, the authoritative DNS server for the domain of the cluster nodes. The address *www.autotalli.com* points to this cluster DNS which upon receiving a request can redirect it to an application server that could have an address like *www.1.autotalli.com*. When the cluster DNS is used, there can be a variety of selection policies which the cluster DNS uses to select the appropriate server. [14]

There is however a problem with the DNS approach which has to do with the limited control on the request reaching the cluster. Because users use the user friendly addresses on the web to navigate there are DNS servers all around the web which map the user friendly addresses to IP addresses. Indeed there are many intermediate name servers between the client and the cluster DNS which can cache a the user friendly name to an IP address to reduce network traffic. [14]

Besides providing the IP address of the cluster DNS the DNS server also specifies a validity period for the user friendly address - IP address mapping. This validity period is called the Time To Live (TTL) for caching the the address. After the TTL the request is forwarded for the cluster DNS for assignment to a server. The worst case scenario is that user is actually using *www.1.autotalli.com* which the cluster DNS selected for this user but from his perspective he sees only *www.autotalli.com*

address. Now if the *www.1.autotalli.com* server becomes inoperative and the TTL has not expired the intermediate DNS servers can still redirect this user to the inoperative server. Fortunately this is a small price to pay and can be mitigated by a waiting a certain time period before actually shutting the server down. [14]

2.6.3 Server-based Approach

A server-based technique to load balancing is to use a two level dispatching mechanism. Initially the clients requests go to the cluster DNS and gets dispatched to one of the application servers. After this the server may choose to respond itself of it can reassign the received request to any other server on the service. This approach aims at solving most of the problems that affect the cluster DNS, such as non-uniform distribution of requests and the limited control over the requests reaching the application servers. [14]

The most efficient way to implement load balancing is to use the server based approach, but given the complex nature of it, the most widely used is the cluster DNS technique. By using cluster DNS the service can fulfill high availability requirements sufficiently and because it is simple method it is mostly used in web services. [14, 15]

2.7 Session replication

Usually complex web services have to use multiple application servers with a load balancer to be able to serve all the users and to distribute the load amongst the servers equally. One application server is running one instance of the application and the user can actually use multiple application servers unknowingly when using the web service. This depends on the configuration of the application servers and the load balancer. If the load balancer uses non-sticky session configuration it means that specific users consecutive requests can be directed to any operative server [16].

The load balancer can also be configured so that it directs the same user to use the same application server behind the load balancer throughout the whole session. This approach is called sticky session configuration [16]. When the application servers use the sticky session load balancing one server has all the information of the users using this particular server stored to its sessions objects. If this instance crashes, is taken out of the load balancer or becomes inoperative some other way all the information stored to this servers sessions are lost.

To avoid such situations and to increase robustness and availability of the web service, session replication can be used. Session replication means that all the application servers are made aware of all the sessions in the service and not just the server specific sessions. The replication or clustering of sessions can be done in multiple different techniques and technologies depending on the services server layout, resources and demands.

In a scenario where a web service has three application servers A, B, C and a load balancer in front of them, the servers can be thought to create a cluster. When a user makes the initial request to the web service, it is first directed to the load balancer which is configured to use sticky sessions. The load balancer directs the user to one of the application servers, server A for example. Now all the requests from this user will go to server A, and the information related to this particular user will be stored to server A.

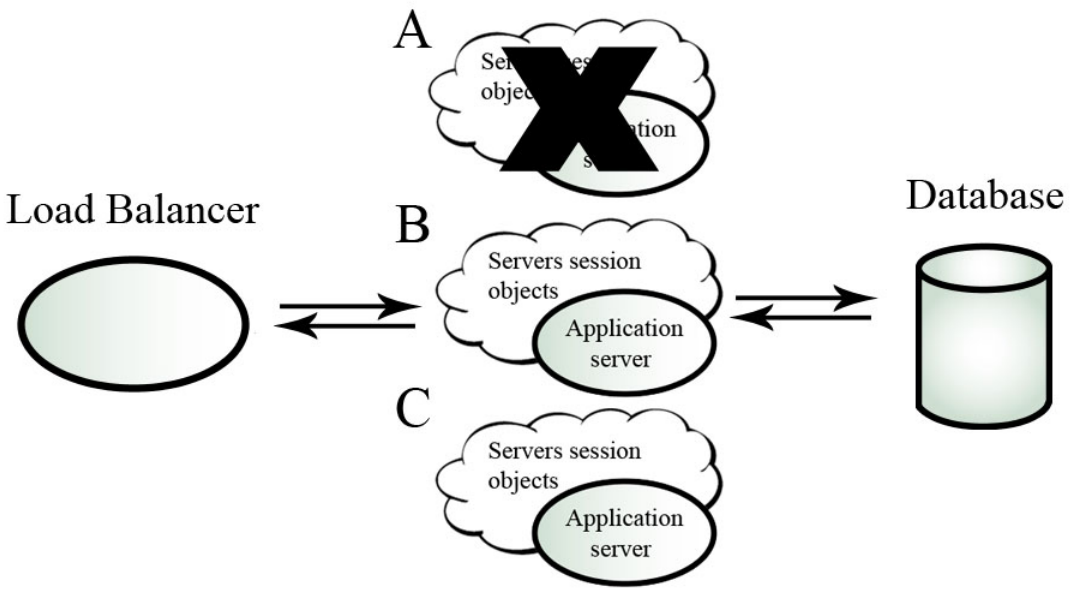

Figure 2.4 *System without session replication.*

Now if server A crashes or becomes inoperative some other way the user is directed to one of the operative servers B and C as illustrated in Figure 2.4. When the users next request arrives to one of the other operative servers they have no previous information about this user and the user will be interpret as a new user. This means a new session will be created for the user on the server. This can be frustrating for the user if he is for example editing some content in the web service during the

server crash and the information he has been creating is lost.

With this kind of setup, where there is no replication, updates and other maintenance work have to be carried out during a time when the user load is minimal in the web service to avoid expiring user sessions. User load is usually lowest during night time so updates have to be carried out at night or early in the morning. Smaller hot fixes that may have to be done after a bigger update also have to wait for the next appropriate time which usually means the next morning.

The above scenario illustrated in Figure 2.4 in a web service that uses session replication is quite different. Because the sessions are stored to a place available for all the servers, all the application servers have knowledge of all the sessions in the service. Now in the scenario where application server A becomes inoperative the user is redirected to another operative server B or C. The server examines its own session objects and when it does not find the user's session in there, it will search the user from the shared session store shown in Figure 2.5. When the server finds the user's session in the store it uses this session for the user. From the user's perspective there was no server crash and the user experience remained intact.

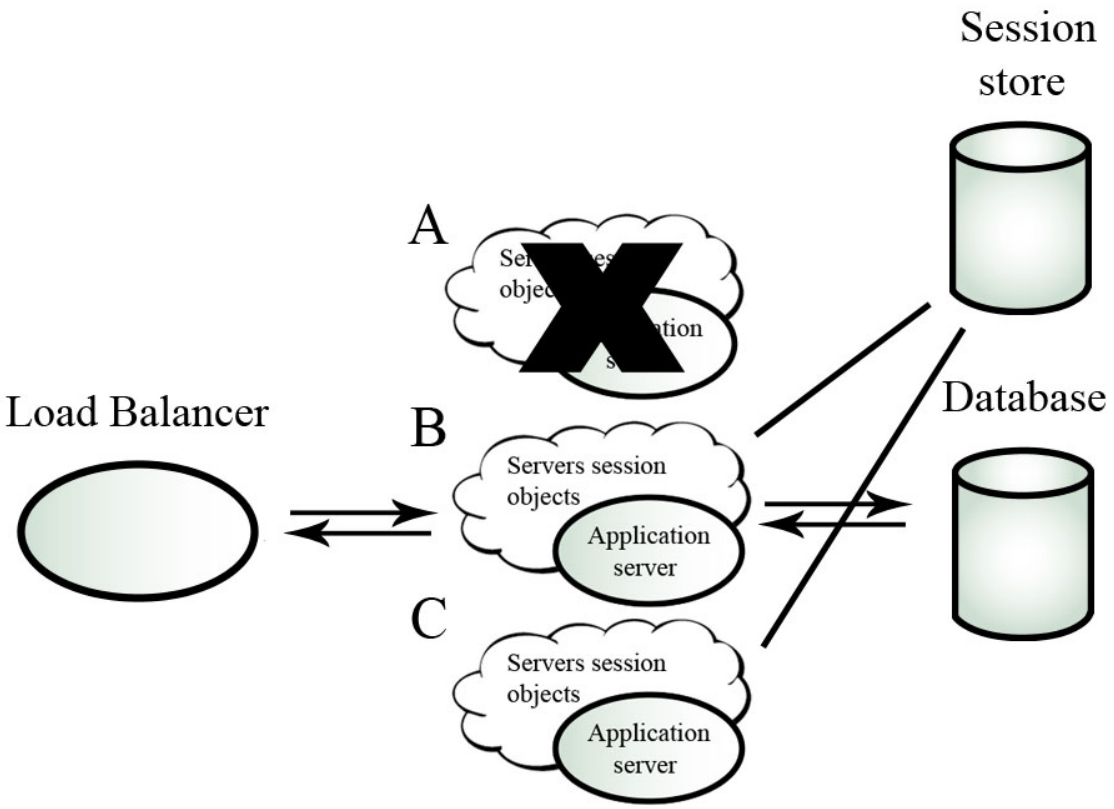

Figure 2.5 *System with session replication.*

Session replication does not only improve web service availability and robustness, but it also improves user experience by reducing the downtime on the servers. By using session replication updates can also be done instantly when the fixes are finished by updating one server at a time instead of waiting for a quieter time in the service and this has a big impact on the web services maintainability.

2.8 Summary

It may seem like it takes only a computer that functions as a server and a computer that functions as the client. However, there are actually a lot of components and techniques that makes browsing the web simple for the user. While technology develops and technologies come and go the concepts introduced in this chapter are the cornerstones of web. Session replication is probably the youngest technology introduced in this chapter, but given the fact that web services are starting to move to different kinds of virtual machines and cloud environments it is going to be a vital technique in the future.

Using session replication increases web services accessibility, stability and maintainability. Individual failures of application servers are no longer a critical problem because the service can function by using the other operative servers while the failed server can be taken care of. Furthermore session replication improves the maintainability because updates can be done immediately after they are finished. Specially in cloud environments session replication improves the scalability of the service.

The downsides of session replication are the increased complexity of the web service and the extra resources needed to run the replication. It may require adding more servers to the service to work efficiently. Session replication can also be an extra vulnerability to the web service because it brings a new point of failure to the system. If the component running the replication becomes inoperative, it could compromise the whole service.

3. REPLICATION TECHNOLOGIES

Using a third party solution to handle sessions instead of using the web service's own application servers brings many advantages. If the session storage is an in-memory based solution, handling of the sessions is much more quicker than using a solution based on a disk storage. The application servers load is also smaller since a third party component is taking care of the sessions. In this chapter we will examine the methods and technologies to which all the replication methods are based on and take a closer look on the probable technologies to be used in the example implementation for this thesis work.

3.1 Replication

Replication is a common technique to ensure accessibility and reliability, and to enforce fault tolerance. Replicating sessions is not the only application area of replication that is used in web services. For example using multiple application servers running instance of the same service is a trivial example of data replication. The disk storage and databases can also be replicated to ensure the safety of the data and to serve as a backup in case of emergency.

There is however a difference when replicating data as a backup and replicating temporary data like sessions. Sessions are changing constantly and they are deleted and accessed more frequently. There are many different ways and technologies to implement session replication and they use many different kinds of approaches to accomplish their goal. Depending on what technology is used the sessions may be stored on hard drive, in a database, in memory or a combination of the previous. Even the actual replication procedure depends on the technology. Sessions could be stored centrally on one big external database to which all the servers have access to or they may be backed up in a distributed manner where one servers session replicas are divided amongst all the other running servers. One thing common to many of the technologies is the use of NoSQL databases.

3.2 NoSQL databases

In a traditional relation based database the database has a well defined schema. This means the structure of data that is stored into the database is predefined by the layout of the tables and fixed types and names of the columns in the tables. This approach is effective when storing structured information that rarely changes such as statistics but not as efficient when storing something with more abstract nature like a sale notice. Every sale notice is different because different fields can be filled with different kind of content. Scaling is a another obstacle for relation based database systems because they do not easily work in a distributed manner and they are not designed to function with data partitioning. Scaling a relational database basically means running it on a bigger, more powerful computer. [17]

Since the nature of the web is distributed the need for distributed storages is obvious. This is where NoSQL databases step in. NoSQL stands for "Not only SQL" and many of the session replication methods make use of a NoSQL database. NoSQL systems generally have six key features in common [18]:

1. Horizontal scaling.
2. Replication ability and partition data over many servers.
3. Simple interface or protocol.
4. Weaker concurrency model then relational databases.
5. Efficient use of distributed indexes and the use of RAM.
6. Ability to dynamically add new attributes to stored data.

NoSQL systems differ from each other on these features, some features are supported more strongly than others but at least some of these characteristics are present on all of the NoSQL databases. In a NoSQL system there is no predefined schema or at least it is really broadly defined. This means that the information that is stored into the database can differ a lot from each other. The data that is stored can be thought as objects. The objects basically have the same type but the contents of the objects can differ a lot so straight comparison is not possible. There are three main different types of NoSQL databases, categorized in accordance to their data model: Key-value stores, Column-oriented databases, and Document oriented databases [17, 18].

NoSQL databases have their advantages and disadvantages compared to relational databases. NoSQL databases enable quicker development because developers can

use the database in a way that suits their needs shaping the stored data as they go. Changes are also quicker to make to an NoSQL database when compared to relation based database where the worst case scenario is dropping the whole schema and creating the whole schema from scratch [17].

3.2.1 Key-Value store

Key-Value stores store data in a map, also known as associative arrays, as their fundamental data model. A key-value store has indexed keys for retrieving the value stored into the map. The key can appear at most once in the collection and data is always stored as key-value pairs, even if the value is empty. [17]

The inside of a key-value map is illustrated in Figure 3.1. The most common case of a key-value pair is a string hash which represent the key and the actual data which is considered as the value in the key-value relationship. The concept of the associate array is the same as in a map offered by programming languages.

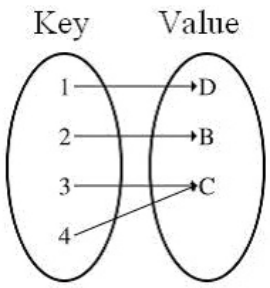

Figure 3.1 *Key-value store map.*

The key-value stores generally provide additional functionality like replication, versioning, transactions and persistence for the data. Redis is a well known open source key-value store [17, 18].

3.2.2 Column-oriented database

Column-oriented databases store data in a similar way that relational databases do where a row represents a data item as illustrated in Figure 3.2(c). The trick is that each of the columns is stored individually in column oriented database as illustrated in Figure 3.2(a) and (b). The difference between column oriented and relation based database is illustrated in Figure 3.2.

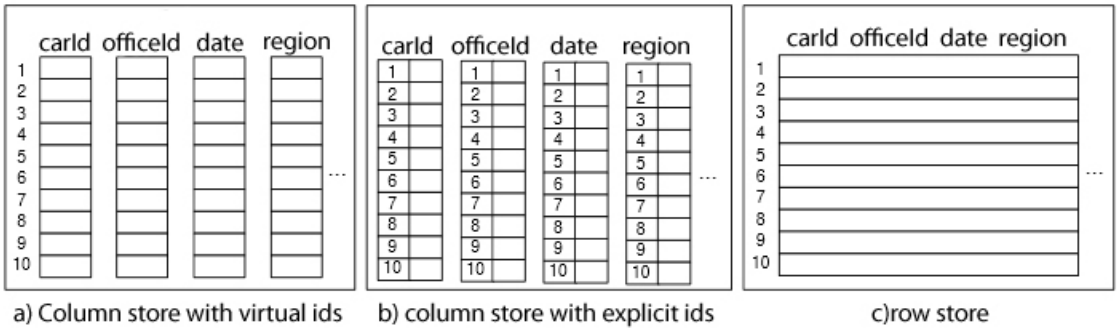

Figure 3.2 *Column oriented storage compared to relational row store.*

Since the columns are stored individually they can be packed really tight because a compression algorithm suitable for the data on the column can be used. An algorithm suitable for integers can be used for integer columns and an algorithm suitable for strings for string columns and so on. Also reading from the database is quick since only the columns needed to make the query are used and not all the columns that are associated with the data item as in relation databases. Apache Cassandra is a good example of a column-oriented database [17, 19].

3.2.3 Document oriented database

As the name says a document oriented database stores data as documents. On a very abstract level a document oriented database could be thought to function similarly as a key-value store. The difference is that document databases support more complex data structures then key-value stores and while they can store documents in the traditional sense like articles, excel files, they can actually store any kind of pointerless objects [17] an is illustrated in Figure 3.3.

Document stores are more complex than key-value stores and they support secondary indexes and multiple types of objects, even nested documents. The size and content of the document item is not predefined so it depends on the user what kind of data is stored into a document store. Document databases are ideal for storing unstructured data and are very flexible because new attributes can be added to documents on the fly. MongoDB is probably the best known example of a document oriented database [17, 18].

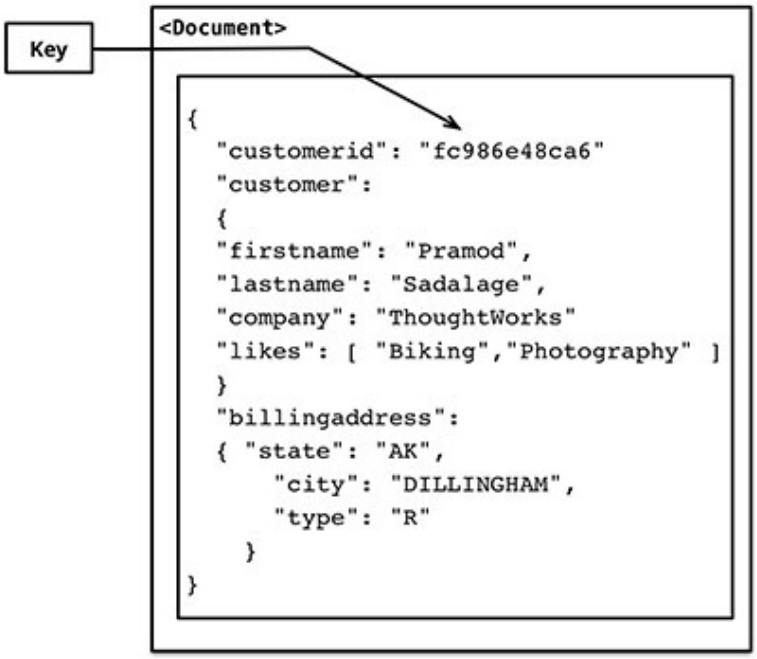

Figure 3.3 *Document oriented database.*

3.3 ACID restraints

ACID stands for Atomicity, consistency, isolation and durability. Atomicity means that an update is performed completely or not at all when manipulating the data. Consistency means that no part of a transaction is allowed to break the rules of the database. So even if a single violation is detected the whole transaction is discarded. Isolation means that each application using the database run their transactions independently. This guarantees that the same result is achieved regardless are the transactions executed concurrently or serially. Durability means that the finished transactions will persist, once finished it can not be undone with a push of a button [17].

In the context of databases ACID means a set of properties that guarantee that the database transactions are processed reliably. In NoSQL databases processing information is generally a lot faster because of the usage of memory and simple data structure. A relational database lacking these strengths also subjects all the data for the same set of ACID restraints, and this slows the processing [17].

The ACID restraints guarantee safer and more sustainable treatment for the data but having to run these constraints on all the data it makes a relational database slower and at the cost of performance and scalability. On the other hand NoSQL databases do not have transactions in the same meaning as relational databases

have so both NoSQL and relation based databases have their own strengths on this matter [17].

3.4 CAP theorem

An inadequacy with NoSQL systems is stated in the CAP theorem. The abbreviation comes from Consistency, Availability and Partition tolerance. A computer scientist by the name of Eric Brewer came up with theorem back in 1998, and it states that it is impossible for a distributed computer system to simultaneously provide all of the three previous guarantees. [20]

Consistency means that all the nodes would hold the same data and give the same answer for a specific request. At best NoSQL systems can guarantee that eventually the nodes will have the same data. Consistency is a difficult subject to handle in distributed environments because there can be numerous nodes and all of them would have to lock and update the data due to be updated simultaneously in order to guarantee that in a specific point in time all the nodes would have the same value for the data. [20]

Availability in CAP theorem context means that the system should always respond eventually. Obviously fast response is better than a slow one but even requiring an eventual response can create problems. In web service's perspective a slow answer is as bad as no response at all given the fact that people do not want to wait for their websites to load for more than few seconds. [20]

Partition tolerance requirement states that the system should remain fully operational even if the system is divided into smaller partitions. Since scalability is one of the NoSQL systems strong features it is still impossible to guarantee that all the nodes communicate with 100% assurance. Messages can be delayed and sometimes lost forever. In environments where the partitioning is automated and backups are assign to nodes dynamically and where the amount of nodes is measured for example by hundreds there can be issues like split brain syndrome. Split brain syndrome is a condition where the data is not divided evenly but the cluster starts to split to pieces where only some nodes communicate to each other and the data is divided between them. This leads to multiple smaller clusters inside the whole cluster [20, 21].

When working in a distributed environments according to the theory only two of these factors can be guaranteed. Examples where different database systems are in the CAP theorem triangle is shown in Figure 3.4. While there is no NoSQL system that could deliver all of them, the quality of the used NoSQL system is always

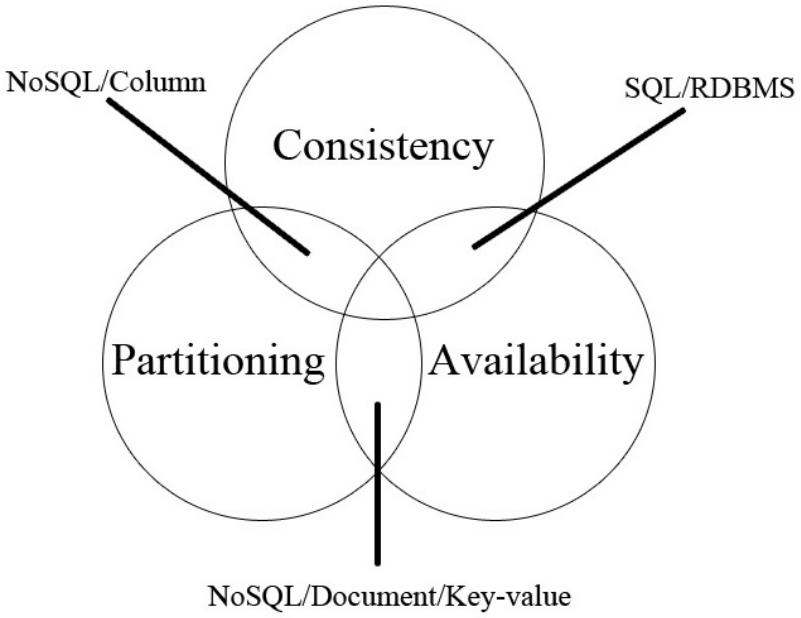

***Figure** 3.4 CAP theorem.*

dependant on the context of the web service it is used in. For example the system might give different responses for two requests made almost at the same time due to requests going to different nodes. The damage is not that big if the data is not critical, lets say how many cars a car portal has but can be vital if the data should always be 100% correct. In smaller systems, consistency can be perfect because of the small number of nodes but there can be availability issues and issues related to partitioning. [20, 21]

The notion these theories make is that there is no perfect solution to any given environment or setup. The selected technologies and methods have to be inline with the goal that is pursued. While traditional databases have different kinds of strengths then NoSQL system they both are strong in their own field. [22, 20, 21]

3.5 Selected session replication technologies

Next we study closer four most promising session replication technologies, Hazelcast, Redis, Cassandra and Memcached. All of these are a bit different in terms of implementation, which should give a good insight to how different technologies comply for the pilot web service.

3.5.1 Hazelcast

Hazelcast is an in-memory data grid written in Java. Hazelcast provides distributed implementations of the java.util.concurrent package and distributed versions of some of the standard java data structures such as Map, List, Set, Queue, and Multi-Map. Hazelcast is an open source project so anyone can use it by simply downloading the library, adding it in their own projects and using it. Hazelcast is designed as lightweight and easy to use and it is delivered in a single JAR file. Apart from the other replication technologies Hazelcast can be more widely used because it is an library and not a data storage. For this reason Hazelcast encourages developers to use the library as a tool to solve concurrent and distribution problems developers are facing during development [23, 24].

Hazelcast provides easily scaling distributed computing which can be considered really fast because the whole data structure is located in memory [23]. In the Hazelcast grid there is no single master node but instead all the nodes in the grid are considered equal to each other. The oldest node is the de facto master but it can be changed on the fly. Every node in the grid stores the same amount of data and every node will take part of the computing equally. The Hazelcast grid is created automatically and the nodes communicate to each other using Multicast or TCP IP protocols. The discovery of the nodes is also automatic [24].

By using Hazelcast it is easy to form grid that has hundreds or even thousands of nodes. The grid can be also formed on a single computer by creating many Hazelcast instances which listen to different ports allowing the use of distributed model on a single machine [25]. Some cloud services such as Amazon deliver their own discovery methods for Hazelcast because the use of Multicast is prohibited in their cloud environments [24].

3.5.2 Redis

Redis is a popular open source in memory key-value store system. Redis is implemented in a very simple way and it does not support complex queries nor complex indexing. Despite this the simplicity is the key factor that makes it one of the fastest in memory key-value stores [26]. To give some kind of perspective Redis is able to execute 100 000 read or write operations in one second [22]. In spite of the simplicity Redis supports multiple data structures and even nested data structures as values. The data structures in Redis are specially called ”Redis data structures”, and they include String, List, Set, Sorted sets, and Hashes. Redis also provides commands to

operate the Redis data structures. For example List supports the familiar operations push and pop [26].

All thought all the data in Redis is kept in memory after a configurable interval the contents of the database are written on disk as a backup. One of the downsides of Redis is its dependency to the available memory [26]. All the operations which Redis makes are atomic in nature which means that they are indivisible and cannot be interrupted. Redis also supports transactions in a way. Transactions in Redis are chained atomic operations and executed in order. If one of the operations fail it does not fail the whole transaction and instead the next operation in the chain is executed. This can affect the consistency of the data in a long run [26].

3.5.3 Cassandra

Apache Cassandra is an open source column-oriented NoSQL database originally developed by Facebook for its own needs. Cassandra data model is a partitioned row store with tunable consistency. Cassandra is designed to handle large amounts of distributed data across servers providing high availability with no single point of failure [22]. It provides linear scaling and asynchronous masterless replication allowing low latency operations for all the clients. [22]

Cassandra's architecture is an distributed database system where all the nodes are equal. Cassandra supports diversely different kind of data structures and has an expressive query language. Even if Cassandra resembles a relation based database, most of the selected technologies it does not support joins or subqueries which derives from the column oriented style it is built. [22]

3.5.4 Memcached

Memcached is an open source, multi-threaded, distributed caching system, which was originally design to temporarily cache data from databases and interfaces in memory which were repeatedly queried [27]. By caching this kind of data memcached helps on reducing service latency and traffic to database. Memcached can be thought as an in-memory key-value store which supports string indices [28]. Memcached is similar to NoSQL stores but it is not persistent which means in the event of failure, all the data stored is lost. So when Memcached is shut down it does not store the data anywhere. To mitigate this threat Memcached is scalable and supports storing data in a distributed manner across multiple servers. Adjunct memory of the servers can be harnessed to be used by Memcached to store data [28, 29, 30].

Memcached has four main data structures: a Hash table to locating the cache item, Least Recently Used list, the cache item and a slab allocator. The hash table data structure is an array of buckets and the arrays size is always a power of 2 to make finding the correct bucket where the searched cache item is quicker. The least recently used list is used to determine the eviction order in which the oldest cache items are removed if necessary. The cache item is the key-value structure which has been introduced already. Lastly the slab allocator provides memory management for the cache items. The cache items are suppose to be relatively small so freeing these small chunks of memory with system calls would be slow and would cause thrashing. [27, 29, 30]

Memcached operates on shared memory basis from which multiple clients can retrieve and store information. Memcached has support for client programs on variety of programming languages which makes it diverse and flexible solution for different kinds of systems. [27, 29, 30]

3.6 Comparison of the selected technologies

The technologies introduced in the previous subsections are all a bit different when compared to each other. All of them operate a bit differently and they all have their advantages and disadvantages. In the beginning of this work Hazelcast seemed like the perfect technology so it was studied the most. Hazelcast combines the caching principles of Memcached to a NoSQL store without actually being a NoSQL store or caching system. One of Hazelcast strengths is its clustering capabilities and the ease of use. The session replication worked with it out of the box and not having to worry about the nodes seemed to make it very easy to use. Hazelcast also provided a very good management center to monitor and configure the data grid.

There was however one big downside with Hazelcast. Despite of the open source license and all, using the management center was not free. The charge on the management center could add up to thousands of dollars in a year when used in production environment with multiple nodes. Searching the web did not provide any alternatives for a realistic open source based management centers for Hazelcast. Managing the data grid would be crucial in production environment if something goes wrong so deploying Hazelcast without the management center was not an option.

The next subject of research was Cassandra. Cassandra is developed by Apache which also developes the Tomcat Web Server, on which autotalli.com is running, so the support seemed good for the existing application. Setting Cassandra up is

simple. First you need to download DataStax Community Edition Apache Cassandra and install it. After the installation the cassandra jars need to be included in the web application and the application needs to be configured to use a cassandra valve and manager to store sessions. An open source manager for Tomcat is available at: *https://code.google.com/a/apache-extras.org/p/tomcat-cassandra*. Cassandra's operating model just was not the perfect option to be used in this case. Cassandra operates by running this third party store to which application servers are connected. Using just one colossal store was not the best option nor the point of this work so the research moved on to the other options. Therefore, Cassandra was also abandoned from the implementation phase.

Redis seemed to be really promising technology and it was the second technology used to implement the session replication with after Hazelcast. Redis has a diversity of client programs and the replication worked well with it but the lack of support for the Apache Tomcat server degraded its chances. When studied more closely the open source session manager which was used used had widely known bugs but no one developing it regularly. The latest commit to the repository was made over two years ago and the session manager had a critical race condition bug that is still unsolved. At the end of the year 2014 the development of the session manager was continued by the open source community. The bug was reported to appear once in every ten thousand times but when Redis was tested these race conditions appeared frequently which basically knocked Redis out.

The next technology which was studied was Memcached. The easy setup and the lightweight implementation made it the best candidate to be used as the actual pilot implementation for autotalli.com application. The deployment was simple and there were actual choice on the matter to what kind of monitoring system to use. When tested no critical issues were encountered. Memcached also has a good fail safe. The sessions are kept in the JVM of the Tomcat's running the application and the sessions are backed up to Memcached so in a case of total crash the system remains functional.

3.7 Related Studies

The are some studies done involving session replication and session clustering from different angles. Some of the studies focuses on the quality aspects of web services in which session replication is seen as a factor that increases the quality of the web service by improving fault tolerance and providing high availability. Also the big data concept relates closely to the technologies introduced [31].

3.7.1 Session Replication as a Quality of Service Factor

In their article EdgeComputing: Extending Enterprise Applications to the Edge of the internet Davis, Parikh and Weihl study a deployment model called EdgeComputing which enables simple application deployment to cloud environments. The article is not focused on session replication but in their studies Davis et. al have implemented an in-memory session replication to provide higher level of fault tolerance to the application. [32]

Moser, Melliar-Smith and Zhao describe in their article Making Web Services Dependable how Web Services create opportunities for efficient ecosystems of consumers and suppliers, collaborating and competing for products and services over the internet. For this reason it is crucial that the infrastructure is dependable and Moser et. al describe multiple methods to ensure this naming session replication as one of them. In their article they do not go in detail how the replication is done but instead refer to an in-memory technique for the Tomcat servlet engine. According to Moser et. al session replication is one technique which provides higher level of reliability, availability and consistency. [33]

Rossi and Turrini analyze the impact of component replication in J2EE clusters and they analyze the mechanisms and strategies that can be adopted by J2EE application server clusters. They describe how the replication can affect both data consistency and application semantics and also degrade the servers performance. They have come to to the conclusion that in order to avoid this the server should support advanced clustering techniques such as sub clustering and master-slave replication. They also emphasize that the open source technologies are not on enterprise level at the time the article was written in 2005 when it comes to fail over. [34]

Using virtualization to improve software rejuvenation is a more theoretical article where Andrzejak, Silva and Torres talk about self rejuvenating web services in which session replication is used to preserve the users state when a web server has to be rejuvenated [35].

3.7.2 Big Data and Replication technologies

Big Data is an all inclusive term for any collection of data sets that is so complex and large that it is difficult to process them using traditional data processing applications. Social networking and e-commerce web services in particular are very interested in big data because the services produce huge amount of data that could be useful if processed.

Web services produce massive amounts of data while they are operating and the data is rarely used. In fact the data generation is staggering. 90% of all data is produced in the last two years alone [36]. Things like server log files, user navigation paths and other user interactions on the site could give valuable insight to how users use the website and what are they interested in on the website. To know these facts in advance would give a big business advantage to promote right kind of content and to sell more targeted advertisement. Furthermore the actual content created in the web service could be refined to statistics to serve the users of the web service. [36]

The obstacle that is standing in the way is the sheer amount of data that can not be processed in the traditional way. In the context of big data this huge amount of information is chopped into pieces and distributed to many nodes processing the pieces of the data simultaneously and then brought together to give the result using a programming model known as mapreduce. The idea is simple: divide the large data set to multiple nodes which process their input and gives output. [36, 37]

The idea of mapreduce could be illustrated with playing cards and the goal is to sum all the numbers in the deck while the face cards would be processed as irrelevant data. Playing cards have four suits so we could imagine four nodes as well. In the mapping phase the data would be divided to four and distributed to the nodes. On each node all the face cards would be thrown away and the number cards placed in buckets according to their suits. Next the nodes would divide the buckets so that one node has one suit. Now all that is left is to sum up the cards and inform the sum. While the previous card example is fairly trivial example this kind of method could be used for example to process server logs in real time to study user behaviour models and to monitor the state of the service. [36, 37]

Big data does not have straight connection to session replication but the technologies introduced previously in this chapter are used in the processing flow of big data. The nature of big data is that it consist of both structured and unstructured data in vast quantities which means the storages have to be flexible and the data has to be processed concurrently. The NoSQL stores introduced offer this kind of flexible and scalable storage for the data and the Hazelcast library itself offer means to process data concurrently even though Hadoop is the main technology in Big Data. [36, 37]

3.8 Summary

In this chapter we have taken a closer look under the hood of what makes it possible to replicate sessions. The options are many and they can be used to some much more than just persisting sessions. One important thing is that there is no single way to

implement session replication. The chosen technology and setup depends heavily on the system that the replication is implemented in.

Even though the internet is full of commercial and non-commercial technologies and tools which work out of the box the baseline and the outcome have to be studied carefully to find the most suitable solution. The life cycle of the system has to be considered also because some open source technologies are easy to set up and deploy but the monitoring of the system could become impossible.

4. IMPLEMENTATION

In this chapter we will take a look at the Apache Wicket framework which is used to create the autotalli.com web application, examine the layout and architecture of the web service and introduce technologies used in the application. The measures needed to implement the different technologies are also included in this chapter.

The web service to which the session replication will be implemented is autotalli.com. Autotalli.com is a car portal which focuses on serving the needs of people buying and looking for used and new cars in Finland. Autotalli.com was introduced in the early 2000 and it is created and developed by Alma Mediapartners. The current version is the third installment of autotalli.com since the release of the service and the development is ongoing. There has been no session replication solution used in the production version of autotalli.com prior to this study. The default Tomcat manager is responsible for the session objects in the application. Autotalli.com has over 100 000 users weekly and it is one of the most popular car portals in Finland.

This study is done on autotalli.com but the goal is to find an implementation which can be used in every Alma mediapartners web service. All Alma Mediapartners web services are built with Wicket framework and the similarity of the different web services allows to use of a similar solution built in this study to be used in the other web services.

4.1 Wicket Framework

Apache Wicket is a lightweight component based framework written in Java. It is a web application framework for creating dynamic web applications by using components. The development of Wicket started with Jonathan Locke in 2005, and in 2007 The Apache Foundation took Wicket as a top level development project. [38]

Wicket follows the model of stateful user interface much like Swing but tries to stay stateless as long as possible. Wicket uses only two technologies: Java and HTML. HTML is used to model the views in the application and Java is for the business

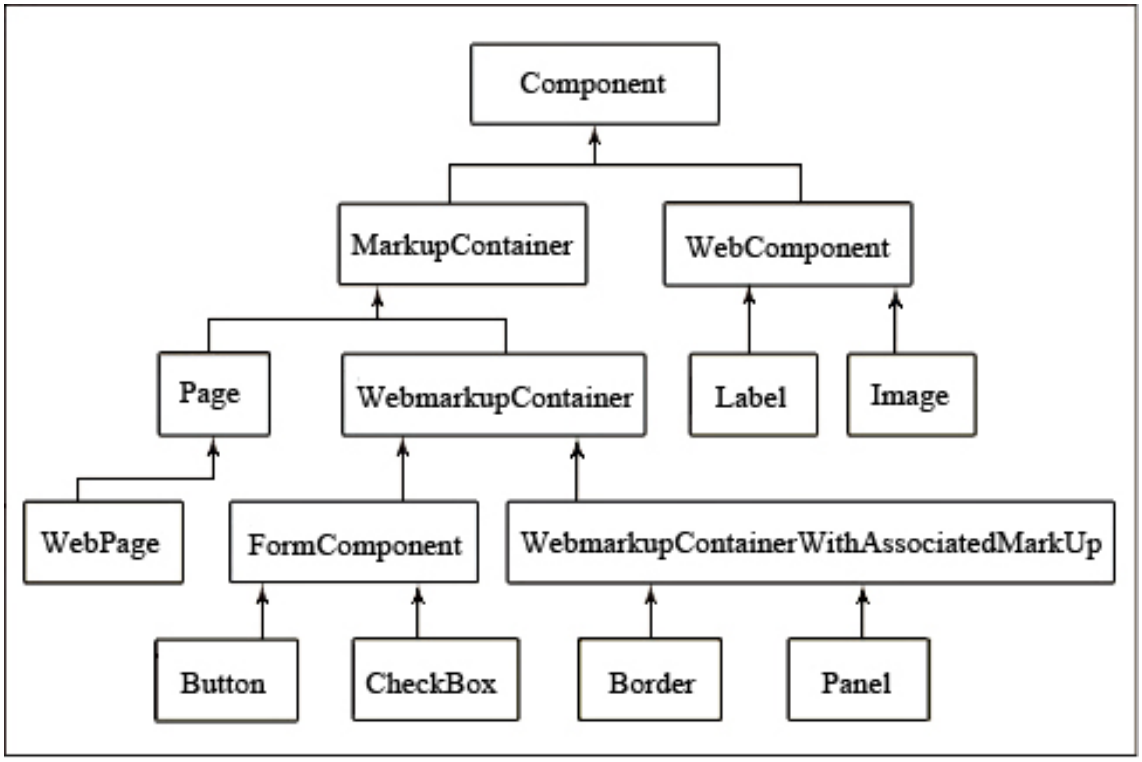

Figure 4.1 *Wicket Component structure.*

logic implementation. HTML templates are used to hold the markup for each element which are bound to named components in the server side java code. The template is responsible for rendering this element in the final output. This way designers can focus only on the presentational HTML side without any knowledge of the implementation logic written in java and the same applies for java developers. [38, 39]

Wicket focuses on object-oriented design and the applications are built with components. Wicket applications form tree-like component sets illustrated in Figure 4.1.

The components such as forms and links use listeners to react to HTTP and AJAX requests. The listener logic is illustrated in Figure 4.2. From Figure 4.2 it can be seen that the browser has a Wicket component with wicket id `AjaxComponent`. When the user interacts with this component the browser makes an ajax request to a path that points to the specific behavior of the component. The server recognizes the component with the id and invokes the correct behaviour, `AjaxBehaviour` seen in Figure 4.2. The server then sends new markup and dynamically generated JavaScript to update or add components via the DOM.

By using Wicket it is easy to build reusable web components that can be extended by

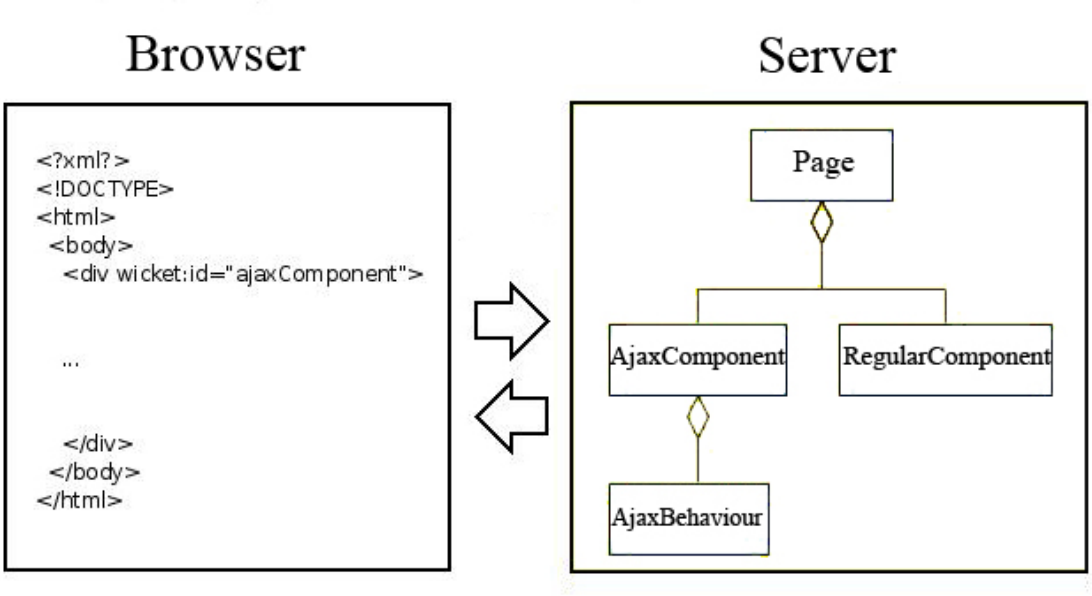

Figure 4.2 *Wicket Listeners.*

using standard inheritance. Every component in Wicket has a model that represents the components state at any given time. [38, 39]

Wicket provides a generic support for web applications. High level web application concepts have corresponding classes in Wicket. For example for the web application Wicket has `WebApplication` class from which there can be only one instance in web application. Wicket provides numerous implementations including session handling, page and logging to name a few. [38, 39]

4.2 Autotalli.com architecture

Autotalli.com uses a variety of technologies and frameworks for various purposes. The application itself is written with Wicket using Java 1.7 [40], and Apache Maven 3 [41] is used project wide for dependency management. Hibernate [42] is used to map tables from a relational database to objects and the actual database runs on PostgreSQL [43]. Spring [44]is used for dependency injection and Apache solr [45] is used as a memory database for search indices and cacheable queries. Various javascript libraries are used to implement front end logic but these technologies are out of the scope of this study.

Autotalli.com web service does not follow any particular architecture model other then layer model and the Wickets principles of design in the web application. The third version of Autotalli was introduced in 2011. Autotalli.com consist of many

services under the hood but users only interact with the web application in which the session replication is to be implemented. Autotalli.com has multiple backline services which can be utilized to hold an instance of the selected technology, for example a Hazelcast node if necessary.

At any given moment during the day there is approximately 300 users in the autotalli.com web service. The livetime for sessions has been configured to 30 minutes and the average session lifetime is 5 minutes. Monitoring shows that there is approximately 7000 alive sessions in the JVM at any given time with the normal user load.

4.2.1 Core Architecture

Autotalli.com web application is built in layered fashion. On the bottom there are the databases Postgres and Solr. The DAO layer is the next layer where Hibernate is used to communicate with the database. The service layer is on top of the DAO layer where Spring is primarily communicating to Hibernate. The line between the DAO and the Service layer is not that clear and they can be thought to create one layer because Spring is used in both service and DAO layers. The layer architecture is illustrated in Figure 4.3

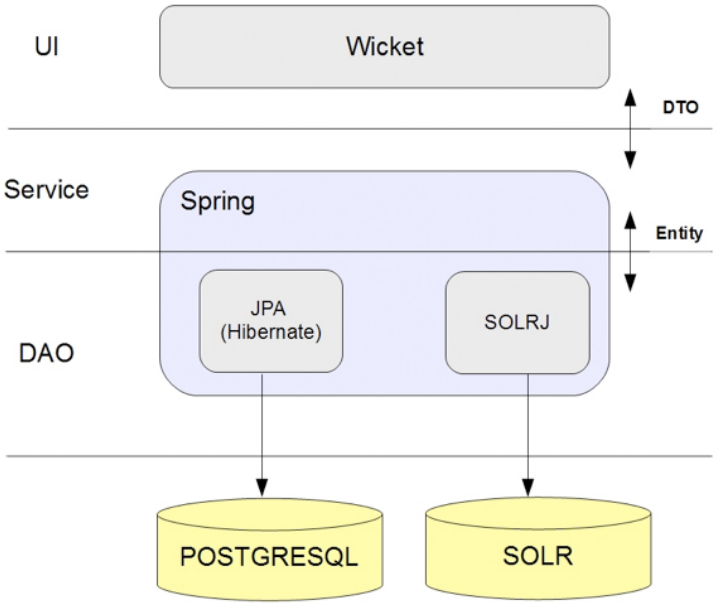

Figure 4.3 *Autotalli - Layer Architecture.*

The Service layer communicates to the UI layer where the Wicket application resides. On the UI layer Spring annotations are used to inject service beans from Spring

application context from the lower layers. Only the Service layer classes are used in UI layer. On the Service layer DAOs and entities are used internally but only DTOs are passed to the UI layer.

Wicket stores the session object and the pagemap associated with the session separately by default. The default Wicket implementation is used in autotalli.com which is a disk storage for the pagemap and the session.

4.2.2 Production server layout

The whole web service runs on four different servers. The database is located alone on one server. Another server holds all the backline services such as internal web service, integrations to other systems and the master memory database. There are two actual web application servers which hold an instance of the web application along with external web services and a slave memory database which updates itself from the master periodically. In front of the web application servers is a DNS load balancer which polls the web application servers every two minutes. The server layout is illustrated in Figure 4.4

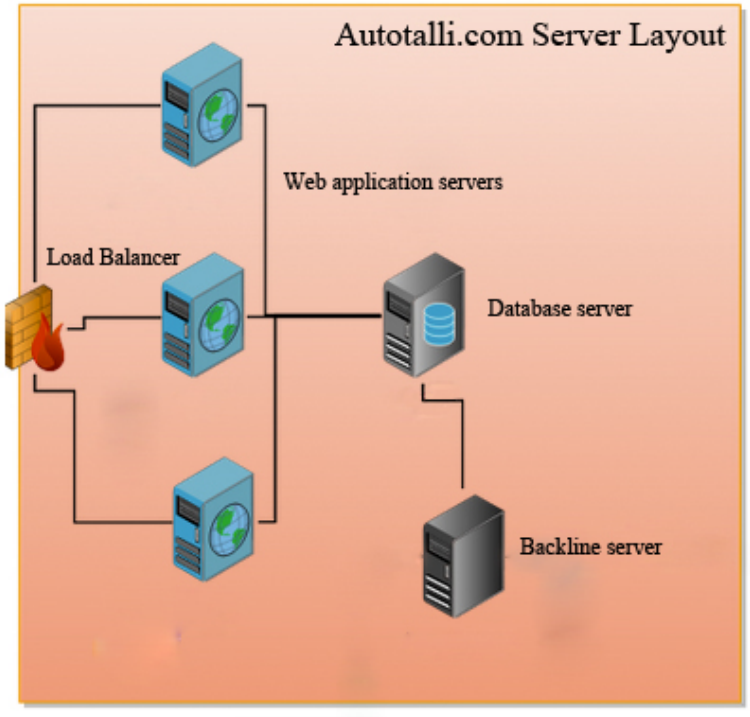

Figure 4.4 *Autotalli.com - Server Layout.*

Depending on the technology the idea is to use the web application servers to hold an instance of the replication node and if necessary deploy a node on the backline server.

Autotalli.com is to be moved to a cloud environment at the beginning of 2015 and this makes these plans somewhat obsolete and the set up for the replication instances has to be planned out for the cloud environment separately. One way is to use a low CPU, high memory server instances which serve only the replication nodes for the web applications servers in the cloud.

4.2.3 Test environment

The testing of the different technologies is to be done on the test environment of autotalli.com which differs from the production environments layout especially from the available resources perspective. In the test environment all the different components reside on one server including the database. The test environment also uses two web applications instances and a load balancer in front of them. There is only one solr slave in the test environment but that does not affect the session replication testing at all.

4.3 Changes made to the application

By default web applications built with Wicket try to be stateless as long as possible so Wickets `WebSession` class has been extended in autotalli to create custom `AutotalliSession` class. The wicket session is not available between request by default but in custom `AutotalliSession` the session has been bound to be available for the whole requestcycle. There are many different attributes stored in the session which we will not be going in more detail but every attribute has to be serializable to avoid causing any problems with the third party session replication stores.

The changes needed to be made in order for the replication to work from the applications perspective is to overwrite the session storage and make sure session attributes are serializable. The default session store in Wicket is disk storage. Overwriting it is fairly straight forward by overwriting the session store with a suitable option that inherits `IDataStore` interface in the class that extends Wickets `WebApplication` as follows:

```
setPageManagerProvider(new DefaultPageManagerProvider()
{
  protected IDataStore newDataStore()
  {
    return  new HttpSessionDataStore(pageManagerContext,
      new IDataStoreEvictionStrategy);
  }
}
```

Wicket offers four different default implementations of `IDataStore` interface: `AsynchronousDataStore`, `DebugDiskDataStore`, `DiskDataStore`, `HttpSessionDataStore`. Anyone can create a custom storage by implementing `IDataStore` interface.

`HttpSessionDataStore` stores everything, the session attributes and the pagemap to the session object. For managing session objects size Wicket offers two `IDataStoreEvictionStrategy` interface implementations: `MemorySizeEvictionStrategy` and `PageNumberEvictionStrategy` for evicting too large session objects. As the names reveal `MemorySizeEviction` evicts old pages from the pagemap based on the size limit set in the instantiation of the strategy and `PageNumberEviction` evicts based on the amount of pages in the pagemap. First in First out is the policy how pages and old data are removed from the session object.

The next step is to get the replication technology up and running, if necessary. At this stage for the different technologies there was different amount of work to be done. Some of the technologies deployed automatically when the application was deployed and others required the store to be already running when the application got deployed. Finally what remained to be done was to ensure everything that is suppose to be stored in the sessionstorage is serializable. Every check, to ensure the session is serializable will not be introduced in the text. With few exceptions all of the data stored in the session was already serializable.

4.3.1 Hazelcast

Using Hazelcast for session replication has fairly low requirements for the working environment. The web application has to be written in Java 1.5 or higher, target web server must support Servlet 2.4 or higher, the session objects that need to be clustered have to be serializable and the web application project needs to have the hazelcast libraries included.

Enabling the session clustering was easy. The target web applications web.xml needs to be alter a bit. The map where the sessions are stored needs to be configured in this file. Also a session listener has to be introduced in the web.xml file.

```
<filter>
    <filter-name>hazelcast-filter</filter-name>
    <filter-class>com.hazelcast.web.WebFilter</filter-class>

    <!-- Name of the distributed map storing
    your web session objects. -->
```

```
    <init-param>
        <param-name>map-name</param-name>
        <param-value>my-sessions</param-value>
    </init-param>
    <!-- How is your load-balancer configured?
    stick-session means all requests of a session is
    routed to the node where the session is first created.
    This is excellent for performance.
    If sticky-session is set to false,
    when a session is updated on a node,
    entry for this session on all other nodes is invalidated.
    You have to know how your load-balancer is configured
    before setting this parameter. Default is true. -->
    <init-param>
        <param-name>sticky-session</param-name>
        <param-value>true</param-value>
    </init-param>

    <!-- Set the debug option. Default is false. -->

    <init-param>
        <param-name>debug</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>hazelcast-filter</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
    <dispatcher>REQUEST</dispatcher>
</filter-mapping>
<listener>
    <listener-class>com.hazelcast.web.SessionListener</listener-class>
</listener>
```

This is the default configuration which can be found from hazelcast.com and from the Hazelcast documentation and it is sufficient as a minimal configuration. Hazelcast provides a variety of options which can be configured and which are not listed in the above configuration. All configurable options can be found from the technical documentation or from the books about Hazelcast [24].

After this the Hazelcast instance is deployed when the web application is deployed and sessions are clustered into the Hazelcast instance which runs on the same JVM, assuming the properties are serializable. When multiple web applications are deplo-

yed the Hazelcast instances discover each other automatically, form a grid and partition the session objects as backups on the other nodes as described in the previous chapter.

4.3.2 Redis

The selected Redis session manager has few requirements for the working environment. The web application has to be deployed on Tomcat 6 or 7. The session manager has release versions for both Java 1.6 and 1.7. Also a few additional java libraries has to be included in the project. The libraries include Jedis, which is a Java Redis client and apache commons pool2 which is a collection of various java helper components.

Setting up Redis as a session store required a bit more work then Hazelcast. Firstly Redis needs to be downloaded and installed in order to get the Redis database up and running. Redis works best on Linux but it also has a version which works on Windows. For my purposes a tried Redis on my work laptop which is running Windows 7. After Redis is up and running a session manager needs to be included and configured for the web application. After the possible options were studied, an open source Redis session manager was selected which can be found from Github: *https://github.com/jcoleman/tomcat-redis-session-manager*. The manager needs to be included into the web application and a Valve and a Manager need to be configured for Tomcat web server in the Tomcat's context.xml file as follows:

```
<Valve className=
        "com.orangefunction.tomcat.redissessions.RedisSessionHandlerValve" />
<Manager className=
        "com.orangefunction.tomcat.redissessions.RedisSessionManager"
         host="localhost"
         port="6379"
         database="0"
         maxInactiveInterval="60"
         sessionPersistPolicies="PERSIST_POLICY_1,PERSIST_POLICY_2,.."
         sentinelMaster="SentinelMasterName"
         sentinels="sentinel-host-1:port,sentinel-host-2:port,.." />
```

After these configurations and preparations when the web application is deployed, assuming Redis is up and running, the sessions are stored in Redis database. With Redis there was actually minor changes needed to be done to `AutotalliSession`. When testing the replication the attributes were not saved to Redis and there was session errors. The reason for this was that Redis session manager did not send the session to be stored if the session was not marked as dirty. By default the session was

not marked as dirty by Wicket when various attributes in the session were modified. This was fixed by marking the session manually dirty when one of its attributes was modified.

4.3.3 Memcached

Setting up Memcached was a lot like setting up Redis. Memcached needs to be downloaded and installed in order to run the store. Memcached has windows distributions binaries to get the development environment up and running on a Windows machine. Setting Memcached up in Linux was a lot easier because Memcached is included with many Linux distributions.

After installing the actual store the application server needs to be configured to use it. The session manager which was used is open source memcached-session-manager which can be found from *https://code.google.com/p/memcached-session-manager/*. The selected manager has flexible configuration, good support for multiple web servers, modifiable serialization strategies and support for cloud environments. The memcached session manager supports Tomcat 6, 7 and 8, handles both sticky and non-sticky sessions and has a failover for both Memcached and Tomcat.

In order to use the Memcached session manager the session manager jar library and spymemcached jar library, which enables monitoring of the cache, needs to be included on the web servers libraries. Optionally if you want to use a custom serialization strategy the library jar for this strategies implementation needs to be included also. A list of ready and supported serializers are listed in the session manager web site. Finally the server needs to be configured to use the manager in the servers context.xml file:

```
<Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
  memcachedNodes="n1:host1.yourdomain.com:11211,n2:host2.yourdomain.com:11211"
  failoverNodes="n1"
  requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"
  transcoderFactoryClass=
        "de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFactory"/>
```

Every Memcached node needs to be introduced in the context.xml file so every server knows the available Memcached nodes. Failover nodes and ignored resources are also introduced in the configuration and the used serializer has to be configured. After this the application can be deployed and the sessions are backed up in Memcached.

4.4 Summary

All of the three replication technologies were straightforward to implement. They also have good wiki pages where instructions can be found how to implement the technology with different types of web server. Wicket web applications are also fairly easy to configure to work with session replication. Most of the work with Wicket can go to examining the session and making sure it is serializable.

5. TESTING AND EVALUATION

In this chapter the test tools and the performance tests are introduced which will be ran against the test environment of autotalli.com. The results of the performance tests for each implemented technology will be described in their own subsections. The different replication technologies are compared to the current implementation where session replication is not used. After the test result there is an evaluation of the implemented technologies. Also the amount of work needed to implement session replication and the added value to the web service are discussed in this chapter.

5.1 Performance tests

The different session replication technologies will be tested in autotalli.com test environment. The test environment runs on one server but it has two application servers and a load balancer. Performance testing will be used to determine how the system performs under higher then normal workload. Performance testing allows to measure the performance and verify other quality attributes such as reliability and resource usage.

5.1.1 Test tools

The performance tests will be conducted with JMeter program which is an Apache project that can be used as a load testing tool for analyzing and measuring performance. JMeter is specifically focused on web applications. The tests will be ran against every technology implemented and the current version where replication is not used to compare the added performance restraints to the system.

Monitoring of the application servers CPU and memory usage will be conducted with Java Management Extensions also known as JMX tools. The test environment allows to monitor only one application server but in this case it is enough.

To supply visual graphs of the tests Java VisualVM is used to monitor the application server while testing. Java VisualVM is a tool that provides graphical interface for

viewing detailed information about java applications while they run in the Java Virtual Machine.

5.1.2 Test Case

The performance test in question is a scenario where 300 threads, which represents users, arrive to autotalli.com frontpage, navigate to used car search form and make a search where the user is taken to the used car listpage. This means there is going to be three requests to the application in each cycle. The ramp up time, which means the time period in which the threads are fired up is set to 30 seconds. This means every second 10 new users are going to arrive to autotalli.com. The usual user load in autotalli.com is 300 so the test will restrain the system on higher then average basis. The test will ran twice so every user will actually make six requests to autotalli.com.

5.2 Test results

The results will be shown in the following named subsections for each of the technologies. In the upper left corner in the figures there is the average CPU load of the server. In the upper right corner is the heap size and how it progresses during the surveillance period. In the lower left corner there is the total number of classes loaded during the whole inspection time and in the lower right corner is the thread count of the process. The threads in this contexts means processing threads and not users as in the test cases context.

5.2.1 Current implementation without session replication

The current autotalli.com with no session replication uses disk pagestorage for the pagemap and keeps the session objects in memory in the JVM. The manager for sessions is the default Tomcat web servers own StandardManager.

From Figure 5.1 it can be clearly seen when the test started as the CPU usage spiked on the application server. The memory usage increased and the heap size was increased to compensate the new load on the server. This amount of users is higher then the normal daily usage in autotalli.com. By examining the results the number of active sessions in the test environment is approximately 300. The active sessions will remain in the JVM for the maximum inactive interval or maximum session lifetime, which is set to 1800 seconds, in the web service.

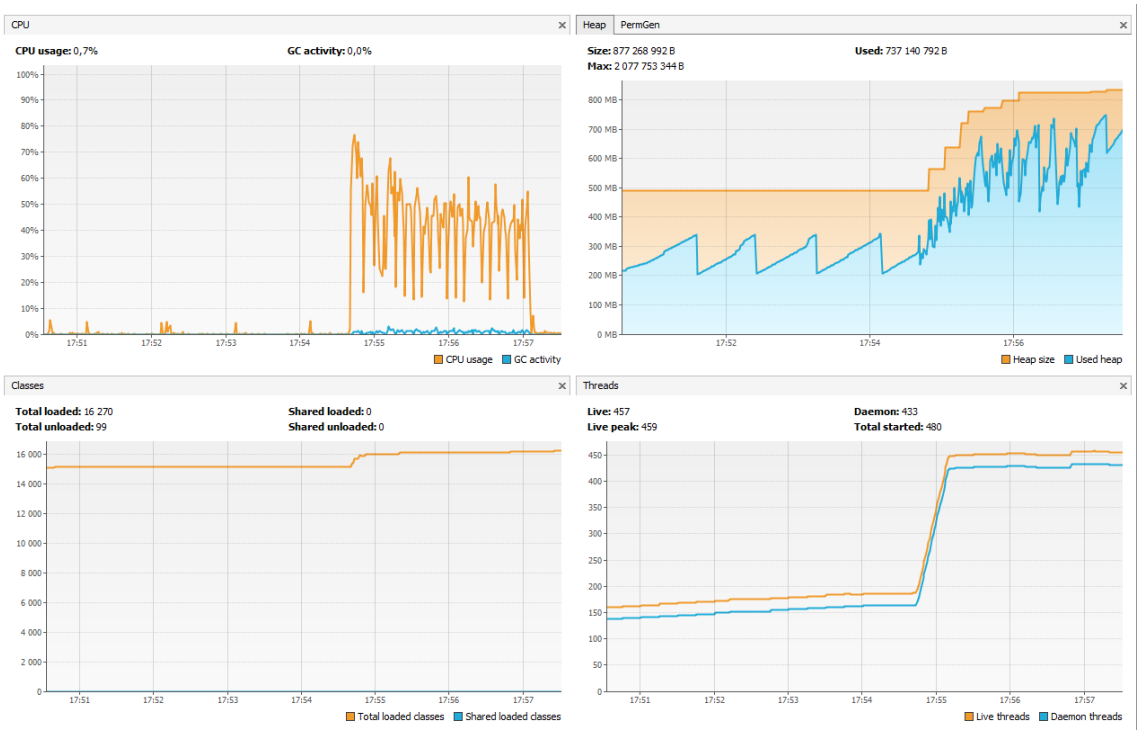

Figure 5.1 *Test environment - No session replication.*

5.2.2 Hazelcast

Hazelcast is the first replication technology tested after the normal version. As Hazelcast backs up the session objects to its own data grid the memory usage seen from the heap size in the upper right corner in Figure 5.2 is almost the same as without replication. This is because the Tomcat's default standardmanager is still handling the sessions.

The memory usage is almost the same comparing Figures 5.1 and 5.2. The serrated pattern which can be seen on the heap size graph is due to garbage collecting which releases resources in intervals.

5.2.3 Redis

The second technology tested is Redis. The resource usage is a bit lower compared to Hazelcast and no session replication which is due to the nature of Redis. The session objects are located only in Redis so the Tomcat's standardmanager has no knowledge of them in its own Java Virtual Machine.

Comparing Figure 5.3 to previous Figures 5.1 and 5.2 the resource usage seems to be lower in every area.

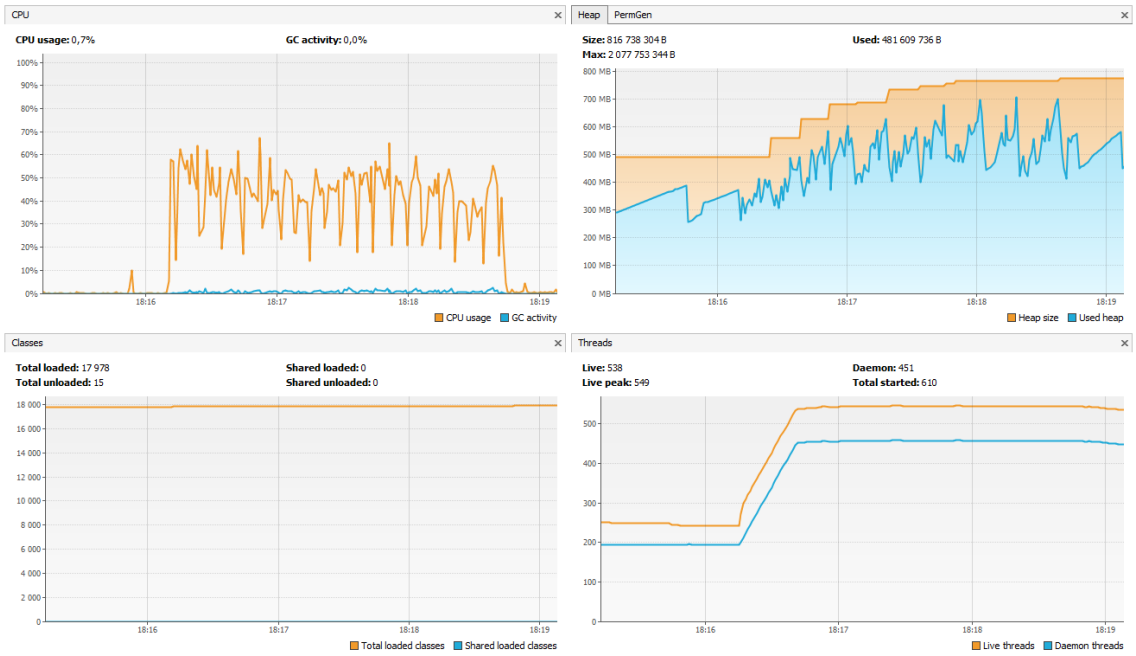

Figure 5.2 *Test environment - Session replication with Hazelcast.*

5.2.4 Memcached

The last replication technology tested was Memcached. It can be seen from Figure 5.4 that resource usage is between Redis and Hazelcast. Memcached uses a bit more heap memory then Hazelcast and Redis but it is on the same level with Redis on threads, classes and CPU usage.

5.3 Evaluation

This section comprises of the evaluation of the results and the success of the pilot implementation. The use of a session replication solution in a web service is also evaluated in a more broader spectrum and the value it adds to a web service is discussed.

5.3.1 Evaluation of the pilot implementation

The pilot implementation done in this thesis work was successful. All three technologies were successfully implemented and the data collected testing these technologies gives valuable insight how the different technologies behave.

The difference between the technologies and the current version considering resource usage is minimal. Running a third party software needed by the technology in

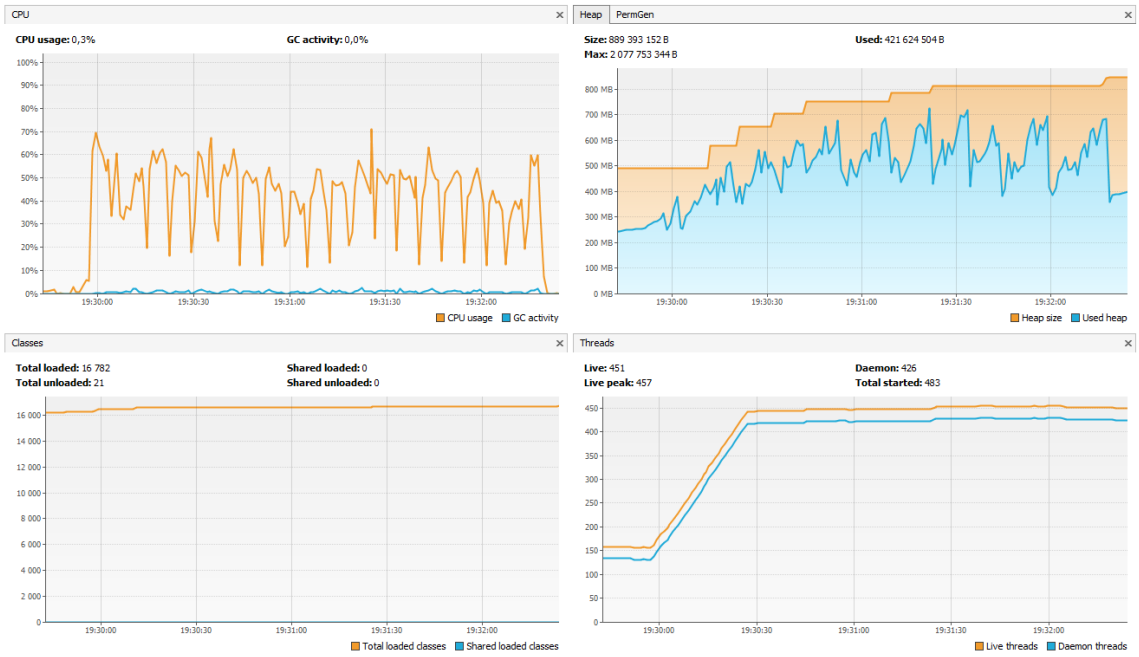

Figure 5.3 *Test environment - Session replication with Redis.*

question requires resources but there was no noticeable difference on autotalli.com applications performance regardless were the session objects stored to a third party store or handled in the Tomcat's on JVM and also backed up to an external storage.

While Redis uses the least resources it presents a new point of failure to the whole system whereas Hazelcast and Memcached do not. The difference in the resource usage is not big enough to support introducing a new point of failure to the system. Overall the characteristics of each technology evens the field when considering the big picture.

5.3.2 Added value to the application

Using session replication increases web services accessibility, stability and maintainability. The added value is obvious when evaluated based on these factors. Service downtime can be removed completely during the smaller updates which require updating only the application servers. In case of application server malfunction users do not lose their sessions when directed to other application servers so the stability and accessibility of the system is increased. Failover of the system is also better. For example using Memcached the application has double backup, since the session objects are kept in the local JVM of the server and they are also backed up in Memcached. This means Memcached can be shut down and the service is functioning normally using the servers local session objects.

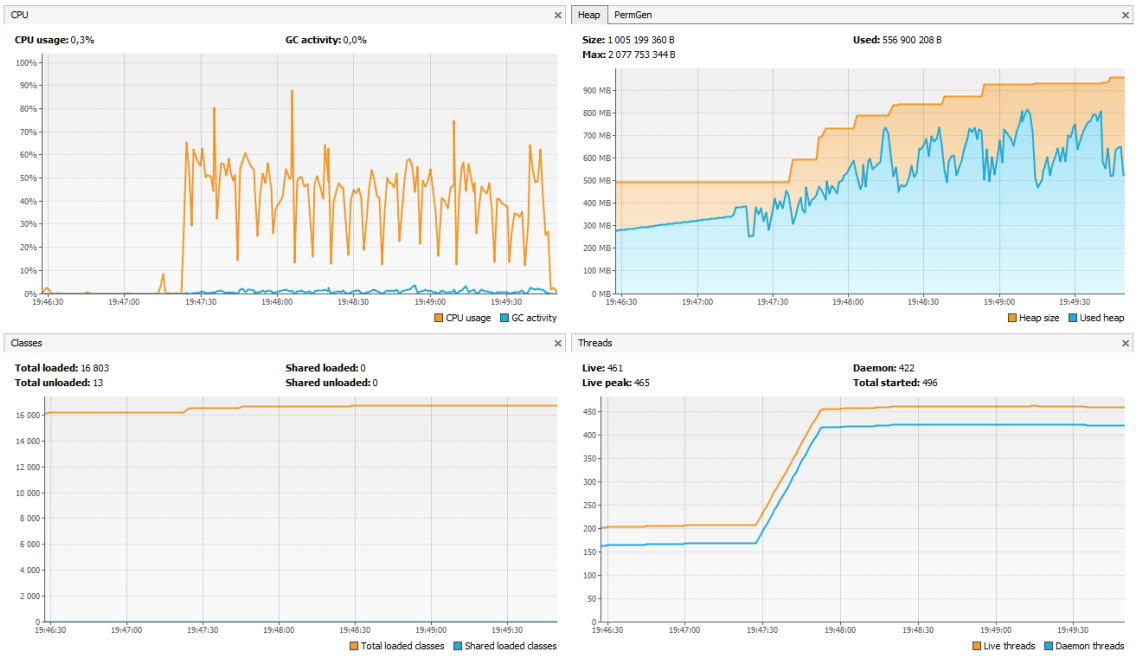

Figure 5.4 *Test environment - Session Replication with Memcached.*

The downside of session replication is the increased complexity of the web service and the extra resources needed to run the replication. From the test results it can be seen that the increased resource usage of the application servers is minimal. Storing the session objects in memory requires a fair amount of memory but that can be reduced by optimizing the sessions size and memory usage. Considering these factors the benefits provided by replication outweigh the disadvantages when talking about a web service the size of autotalli.com.

5.4 Amount of work

The background information needed to implement session replication into an existing web service can be acquired in a fairly short time given that the person has a basic knowledge of the architecture of the web service and some knowledge about the possible technologies which can be used to implement session replication. Because there are multiple open source solutions in the web, it does not take long to find a technology which suits the needs of the web service in question.

The real work is to find a suitable solution which is reliable throughout the whole lifecycle of the web service. The system providing the replication has to be robust with proper monitoring tools and fail safes in place. The fact that many of the technologies work out of the box means more work has to be done examining how the system is working with the web service in question. The layout of the servers

and the resources of the server can play a critical part on what technology can and should be used. Also when working with legacy systems the prerequisites need to taken into consideration.

5.5 Summary

Testing the different technologies was successful and the comparison was easy because of the same test case for each technology including the current web application where replication is not used. The resource use of each replication technology was low and nothing unusual came up from the test results. All the results were encouraging towards the use of session replication.

When considering using session replication there are some facts that need to be thought carefully before jumping into implementing some technology into a web service. First thing that needs to be considered is, will the added value from the replication outweigh the disadvantages brought by the replication. Implementing a replication technology increases the complexity of a web service so if the gain is smaller than the price the necessity of replication in that particular case is questionable.

6. CONCLUSIONS

The goal of this thesis was to investigate session replication technologies and implement a working solution for testing. By this criteria the work done in this thesis was a success. The research on NoSQL technologies was interesting and gives a lot of understanding about databases in general. To conclude this thesis we will discuss how to implement and use session replication to get the most out of it and what needs to be investigated and done in the future considering replication.

Session replication is fairly simple to implement into a web service and in the future it will probably be a regular component in different size of web services. There are multiple different open source technologies which can be used to implement session replication in the internet. What needs to be remembered is that every web service is different and work with different customers so the technology which will be used to implement session replication needs to fit the web services needs. Out of all the technologies researched in this thesis Memcached proved to be the most promising to be used in autotalli.com because of its easy deployment, the ability numerous and powerful monitoring tools and the strong failover capabilities.

In this thesis work a pilot implementation was created for autotalli.com but there are multiple things to enhance and further investigate. The monitoring tool to be used in the production environment needs to be researched and it should be able to give warnings if something unexpected happens. The actual implementation needs to be implemented and fine tuned into the production environment. Also autotalli.com is going to be transferred into Amazons cloud environment so the implementation needs to be configured to work in that environment.

In early 2015, Spring released new session interface which can interact directly with a third party store without the need of a manager so this is something that needs to be studied more in the future.

REFERENCES

[1] H. S. Oluwatosin, "Client-server model," *IOSR Journal of Computer Engineering (IOSR-JCE)*, vol. 16, no. 1, pp. 67–71, 2014.

[2] H. Zhang, "Architecture of network and client-server model," *arXiv preprint arXiv:1307.6665*, 2013.

[3] C. Kambalyal, "3-tier architecture." http://www.keyos-editions.com/commun/exploitation/_catalogues/tmp/seminaires/NTierArchitecture.pdf, 2010. referenced 20.11.2014.

[4] J. Fong and R. Hui, "Application of middleware in the three tier client/server database design methodology," *Journal of the Brazilian Computer Society*, vol. 6, no. 1, pp. 50–64, 1999.

[5] D. Menasce, "Qos issues in web services," *IEEE internet computing*, vol. 6, no. 6, pp. 72–75, 2002.

[6] C. Yue, M. Xie, and H. Wang, "An automatic http cookie management system," *Computer Networks*, vol. 54, no. 13, pp. 2182–2198, 2010.

[7] D. M. Kristol and L. Montulli, "Http state management mechanism," tech. rep., Bell Laboratories, Lucent Technologies, 2000.

[8] J. S. Park and R. Sandhu, "Secure cookies on the web," *IEEE internet computing*, vol. 4, no. 4, pp. 36–44, 2000.

[9] S. Ansari, R. Kohavi, L. Mason, and Z. Zheng, "Integrating e-commerce and data mining: Architecture and challenges," in *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pp. 27–34, IEEE, 2001.

[10] M. S. Ackerman, L. F. Cranor, and J. Reagle, "Privacy in e-commerce: examining user scenarios and privacy preferences," in *Proceedings of the 1st ACM Conference on Electronic Commerce*, pp. 1–8, ACM, 1999.

[11] H. Chen and P. Mohapatra, "Session-based overload control in qos-aware web servers," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, pp. 516–524, IEEE, 2002.

[12] A. D. Miyazaki, "Online privacy and the disclosure of cookie use: Effects on consumer trust and anticipated patronage," *Journal of Public Policy & Marketing*, vol. 27, no. 1, pp. 19–33, 2008.

[13] R. Gill, J. Smith, and A. Clark, "Experiences in passively detecting session hijacking attacks in ieee 802.11 networks," in *Proceedings of the 2006 Australasian Workshops on Grid Computing and E-Research-Volume 54*, pp. 221–230, Australian Computer Society, Inc., 2006.

[14] V. Cardellini, M. Colajanni, and S. Y. Philip, "Dynamic load balancing on web-server systems," *IEEE Internet computing*, vol. 3, no. 3, pp. 28–39, 1999.

[15] H. Bryhni, E. Klovning, and O. Kure, "A comparison of load balancing techniques for scalable web servers," *Network, IEEE*, vol. 14, no. 4, pp. 58–64, 2000.

[16] F. Lu, S. Parkin, and G. Morgan, "Load balancing for massively multiplayer online games," in *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, p. 1, ACM, 2006.

[17] N. Leavitt, "Will nosql databases live up to their promise?," *Computer*, vol. 43, no. 2, pp. 12–14, 2010.

[18] R. Cattell, "Scalable sql and nosql data stores," *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2011.

[19] D. Abadi, P. A. Boncz, S. Harizopoulos, S. Idreos, and S. Madden, "The design and implementation of modern column-oriented database systems," *Foundations and Trends in Databases*, vol. 5, no. 3, pp. 197–280, 2013.

[20] S. Gilbert and N. A. Lynch, "Perspectives on the cap theorem," Institute of Electrical and Electronics Engineers, 2012.

[21] E. Brewer, "Cap twelve years later: How the"rules"have changed," *Computer*, vol. 45, no. 2, pp. 23–29, 2012.

[22] J. Han, E. Haihong, G. Le, and J. Du, "Survey on nosql database," in *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, pp. 363–366, IEEE, 2011.

[23] P. Kathiravelu and L. Veiga, "An elastic middleware platform for concurrent and distributed cloud and map-reduce simulation-as-a-service," tech. rep., INESC-ID, 2014.

[24] M. Johns, *Getting Started with Hazelcast*. Packt Publishing Ltd, 2013.

[25] P. Kathiravelu and L. Veiga, "Concurrent and distributed cloudsim simulations," tech. rep., Mascots, 2014.

[26] M. Paksula, "Persisting objects in redis key-value database," *White paper, http://www. docstoc. com/docs/91018477/Persisting-Objects-in-Redis-Key-Value-Database*, 2010.

[27] J. Petrovic, "Using memcached for data distribution in industrial environment.," in *ICONS*, pp. 368–372, 2008.

[28] J. Jose, H. Subramoni, M. Luo, M. Zhang, J. Huang, M. Wasi-ur Rahman, N. S. Islam, X. Ouyang, H. Wang, S. Sur, *et al.*, "Memcached design on high performance rdma capable interconnects," in *Parallel Processing (ICPP), 2011 International Conference on*, pp. 743–752, IEEE, 2011.

[29] S. R. Chalamalasetti, K. Lim, M. Wright, A. AuYoung, P. Ranganathan, and M. Margala, "An fpga memcached appliance," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, pp. 245–254, ACM, 2013.

[30] A. Wiggins and J. Langston, "Enhancing the scalability of memcached," *Intel document, unpublished*, 2012.

[31] A. Moniruzzaman and S. A. Hossain, "Nosql database: New era of databases for big data analytics-classification, characteristics and comparison," *arXiv preprint arXiv:1307.0191*, 2013.

[32] A. Davis, J. Parikh, and W. E. Weihl, "Edgecomputing: extending enterprise applications to the edge of the internet," in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pp. 180–187, ACM, 2004.

[33] L. E. Moser, P. M. Melliar-Smith, and W. Zhao, "Making web services dependable," in *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on*, pp. 9–pp, IEEE, 2006.

[34] D. Rossi and E. Turrini, "Analyzing the impact of components replication in high available j2ee clusters," in *Autonomic and Autonomous Systems and International Conference on Networking and Services, 2005. ICAS-ICNS 2005. Joint International Conference on*, pp. 56–56, IEEE, 2005.

[35] L. Moura Silva, J. Alonso, P. Silva, J. Torres, and A. Andrzejak, "Using virtualization to improve software rejuvenation," in *Network Computing and Applications, 2007. NCA 2007. Sixth IEEE International Symposium on*, pp. 33–44, IEEE, 2007.

[36] S. A. Pardeshi and P. K. Akulwar, "Escalation the power of big data," NCI2TM, 2014.

[37] P. Sridhar and N. Dharmaji, "A comparative study on how big data is scaling business intelligence and analytics," *Int. J. Enhanced Res. Sci. Technol. Eng*, vol. 2, no. 8, pp. 87–96, 2013.

[38] M. Dashorst and E. Hillenius, *Wicket in Action*. Greenwich, CT, USA: Manning Publications Co., 2008.

[39] V. Okanovic, "Web application development with component frameworks," *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on*, pp. 889–892, 2014.

[40] Oracle Corporation, "Java platform standard edition 7 documentation." http://docs.oracle.com/javase/7/docs/. referenced 20.1.2015.

[41] Apache Foundation, "Apache maven 3 documentation." http://maven.apache.org/guides/index.html. referenced 14.1.2015.

[42] Red Hat, Inc., "Hibernate orm documentation." http://hibernate.org/orm/documentation/. referenced 14.1.2015.

[43] The PostgreSQL Global Development Group, "Postgre sql documentation." http://www.postgresql.org/docs/. referenced 14.1.2015.

[44] Pivotal Software, Inc., "Spring documentation." https://spring.io/docs. referenced 14.1.2015.

[45] Apache Foundation, "Apache solr documentation." http://lucene.apache.org/solr/. referenced 14.1.2015.